Auction-based Admission Control for Continuous Queries in a Multi-Tenant DSMS

Lory Al Moakar [1‡], Panos K. Chrysanthis [1], Christine Chung [2‡], Shenoda Guirguis [1], Alexandros Labrinidis [1], Panayiotis Neophytou [1] and Kirk Pruhs [1]

[1] Department of Computer Science, University of Pittsburgh

[2] Department of Computer Science, Connecticut College

The growing popularity of monitoring applications and "Big Data" analytics used by a variety of users will lead to a multi-tenant data stream management system. This paper deals with the problem of admission control of continuous queries, where the stream processing resources are sold to the end users. We employ variable pricing by means of auction-based mechanisms. The admission control auction mechanism determines which queries to admit, and how much to charge the user for each query in a way that maximizes system revenue. The admission mechanism is required to be *strategyproof* and *sybil-immune*, incentivizing users to use the system honestly. *Specifically, we require that each user maximizes her payoff by bidding her true value of having a query run.* We further consider the requirement that the mechanism be sybil-immune: that is, no user can increase her payoff by submitting queries that she does not value. Given the above requirements, the main challenges come from the difficulty of effectively utilizing shared processing of continuous queries. We design several payment mechanisms and experimentally evaluate them.

Keywords: Data Streams, Multi-Tenant, Admission Control, Game Theory, Auction Mechanisms, Strategyproof, Sybil-immune

1. INTRODUCTION

The growing need for *monitoring applications* such as the real-time detection of disease outbreaks, tracking the stock market, environmental monitoring via sensor networks, and personalized and customized Web alerts, has led to a paradigm shift in data processing paradigms, from Database Management Systems (DBMSs) to Data Stream Management Systems (DSMSs) (e.g., [Abadi et al. 2003; Arasu et al. 2003; Chandrasekaran et al. 2003; Chakravarthy and Jiang 2009; Sharaf et al. 2008]). In contrast to DBMSs in which data is stored, in DSMSs, monitoring applications register Continuous Queries (CQs) which continuously process unbounded data streams looking for data that represent events of interest to the end-user.

There are already a number of commercial stand-alone DSMSs on the market, such as Streambase [Streambase 2006], IBM System S [System S 2008], Coral8 [Coral8 2004], Esper [Esper Tech 2006] and MS StreamInsight [Microsoft StreamInSight 2008] aiming to support specific applications.

We firmly believe that the growing need for monitoring applications and "Big Data" analytics by a variety of users, will inevitably *establish a need for a multi-tenant data stream management system*, and expect the issues relating to outsourcing these services to end users to be particularly important and challenging.

One such challenge is the admission/pricing of continuous queries submitted by end-users. Auctions, used for example by Google to sell search engine ad words, are a proven way to both

‡ Authors are listed in alphabetical order. This work was done while the third author was a graduate student at the University of Pittsburgh.

Authors' addresses: Lory Al Moakar, Panos K. Chrysanthis, Alexandros Labrinidis, Panayiotis Neophytou and Kirk Pruhs, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, U.S.A.; Christine Chung, Computer Science Department, Connecticut College, New London, CT, U.S.A.; Shenoda Guirguis, Intel co, Hillsboro, OR, U.S.A..

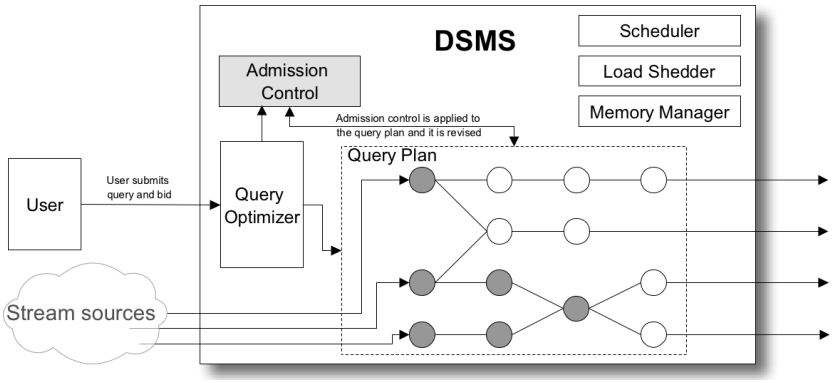


Figure 1: A DSMS architecture. In the query plan the operators in gray are share operators between one or more queries.

maximize a system's potential profit, as well as to appeal to the end-user (client). Instead of a business selling their services at a set price, an auction mechanism (soliciting bids, then selecting winners) allows a system to charge prices per client based on what the individual client is willing to pay. Those who do not get serviced are not denied arbitrarily due to the system's limited resources, but instead feel legitimately excluded because their bid was simply not high enough. And perhaps most compellingly, an auction setting allows the system to subtly control the balance between overloading their servers and charging the right prices.

In this paper[1], we investigate auction-based mechanisms for admission control and variable pricing of CQs to be serviced by the multi-tenant DSMS. To the best of our knowledge, auction-based CQ admission control in DSMSs has not been investigated in the literature. Interestingly, Amazon Elastic Computing Cloud (EC2) [AmazonEC 2009], which utilizes auction mechanisms to sell capacity, appeared shortly after our initial results were accepted for publication in ICDE 2010 [Al Moakar et al. 2010]. This confirms that our vision was timely and valid.

The typical architecture of a DSMS is depicted in Figure 1. In such a system, users submit queries to the system, and the query optimizer generates an operator network. This network takes advantage of shared processing (gray operators in Figure 1 are shared among multiple queries).

In this work we describe a set of policies to be applied at the Query Admission Control component, that determines which queries to admit, and how much to charge the users for each query in a way that maximizes system revenue. Note that in our work, admission control happens at the query level and not at the tuple level as is the case of load shedding algorithms in DSMSs [Gedik et al. 2005][Tatbul et al. 2003], which aim at handling overload situations.

Challenges. One of the key challenges in the design of these auction mechanisms is to determine how to best take advantage of the shared processing between CQs. The fact that some queries can share resources obfuscates each query's actual load on the system. Without clear-cut knowledge of each query's load on the system, optimally selecting the queries to admit becomes exceedingly challenging from a combinatorial perspective.

From a business point of view, the most obvious design goal for the admission control mechanism is to maximize profit. Another first class design goal for the mechanism is to not be manipulable by users. Specifically, we desire that the mechanism be *strategyproof* (also known as *incentive compatible* or *truthful*), which means a client always maximizes her payoff by bidding her true valuation for having her query run, regardless of what the other clients bid. Auction-based profit-driven businesses like eBay and Google AdWords attempt to design and use strategyproof auction mechanisms, even at the expense of potential short-term profit, because when users per-

[1]A four page version of this paper containing initial results of this work appeared in ICDE 2010 [Al Moakar et al. 2010].

ceive that the system is manipulable, they have less trust in the system and are less likely to continue using it. Hence, requiring that the auction based admission control mechanisms be strategyproof is an investment in the long-term success.

Another way users may manipulate the system, besides not being truthful about their valuations, is by submitting bogus queries. Specifically, a user may increase her payoff by submitting queries that she has no interest in, in order to "fool" the system into believing the queries are of high value. Formally this user behavior is characterized as a Sybil attack [Douceur 2002]. We call a mechanism that is not susceptible to this kind of manipulation *sybil-immune* (also known as false-name proof [Sakurai et al. 1999]). Hence, toward establishing the DSMS center, our ultimate goal is to design a CQ admission control mechanism that is strategyproof and sybil immune.

In this paper, we develop a number of admission control mechanisms, and to evaluate whether these mechanisms are strategyproof and/or sybil-immune. We have also experimentally identified potential tradeoffs in terms of system profit, client payoff and rate of CQ admission. Clearly the most important of these properties, from the service provider's point of view, is system profit. Interestingly, the mechanism that is strategyproof and sybil immune offers the best tradeoff with respect to profit.

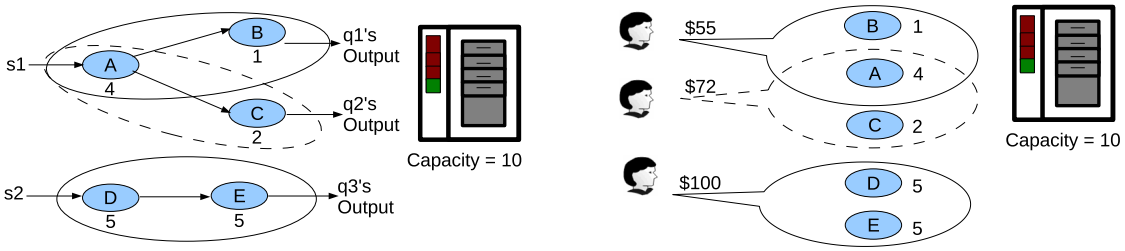
Our Contributions. To summarize, our contributions are:

- We apply techniques and principles from algorithmic game theory to a data streams query admission control problem.
- We propose multiple pricing mechanisms for a multi-tenant DSMS which adapt to the changes in the market demand.
- We propose a number of mechanisms for this problem (four natural, greedy mechanisms and one randomized mechanism) and show that they are *strategyproof*, but only one, called CAT (CQ Admission based on Total load), is also sybil immune.
- Our proposed mechanisms support shared processing of continuous queries and take that into consideration to achieve sybil-immunity and strategyproofness.
- We experimentally show that greedy mechanisms (which take into account both the bid and the load for each user's query), provide both increased system profits as well as better total user payoff, compared to a randomized algorithm that has a profit guarantee. In particular, CAT provides the best tradeoff with respect to profit.

Road map. We define the system model in Section 2 and provide a summary of the relevant background in Section 3. We present our proposed mechanisms and prove their strategyproofness in Section 4 and analyze their sybil immunity in Section 5. We evaluate the proposed mechanisms experimentally in Section 6. We discuss extensions of our problem in Section 7. Finally, we conclude in Section 8.

2. SYSTEM MODEL

A continuous query evaluation plan can be conceptualized as a data flow tree [Carney et al. 2002],[Babcock et al. 2003], where the nodes are operators that process tuples and edges represent the flow of tuples from one operator to another. An edge from operator o_i to o_j means that the output of o_i is an input to o_j. For example, the edge from A to B in Figure 2(a) means that the operator B reads the output of operator A. Each operator is associated with a *queue* where input tuples are buffered until they are processed. We assume an underlying query model similar to the Aurora model [Abadi et al. 2003] where operator subnetworks are connected via connection points that allow the dynamic reconfiguration of the continuous query evaluation plan. At the beginning of a reconfiguration, existing tuples are stored at the connection points and at the end of the reconfiguration, these tuples are input into the new subnetwork before the new arriving tuples are processed. For our purposes, it is sufficient to view a CQ as a collection of operators ignoring their dependencies (Figure 2(b)).



(a) Query Plan. Each operator is labeled with the load associated with it.

(b) Simplified plan, omitting operators order information. User bids are shown as dollar amounts.

Figure 2: Queries in Example 1 as seen by the DSMS.

In our data center (Figure 1), the DSMS has an admission control mechanism that supports a subscription period. During each subscription period, say a day, users submit queries q_i ($i = 1$ ... n) along with bids b_i. At the end of each subscription period, the admission control mechanism evaluates the users' bids and relevant information about their CQs, and returns a decision about which CQs to admit and run the next day[2]. The mechanism also returns the price p_i charged to each CQ_i that is admitted. After each user is charged a price, she cannot withdraw her query.

A bid b_i expresses a declared bound on how much a user is willing to pay to have query q_i executed. Further each user has a *private value* v_i expressing how much having query q_i run is really worth to her. The *payoff* (aka utility) u_i of the user that submitted query q_i is $v_i - p_i$ if q_i is accepted, and 0 otherwise.

We assume that each operator o_j has an associated load c_j that represents the fraction of the system's capacity that o_j will use, and this load can at least be reasonably approximated by the system [Abadi et al. 2003; Sharaf et al. 2008]. The aggregate load of the operators in the accepted CQs can be at most the capacity of the server. We model the system capacity as the amount of work that can be executed in a time unit, given the system's resources (CPU, memory, etc.).

Multiple queries with common sub-expressions are usually merged together to eliminate the repetition of similar operations [Sellis 1988]. For this reason we consider two different definitions of the load of the query. The *total load* and the *fair share load* of a query, defined as follows.

DEFINITION 1. (Total Load C_i^T) *The* total load C_i^T *of query* q_i *is equal to the sum of the loads of all the operators of* q_i.

DEFINITION 2. (Fair Share Load C_i^T) *The* fair share load C_i^F *of query* q_i *is equal to the sum of the fair-share load of its operators, where the* fair-share load *of an operator is the load of the operator divided by the number of queries that share that operator.*

It is expected that many CQs may contain the same operator. Shared operator processing has already been proposed and utilized in the literature ([Abadi et al. 2003; Madden et al. 2002; Krishnamurthy et al. 2006]). Operator sharing is based on the premise that many CQs are monitoring a few hot streams, and many of the CQs are similar, but not identical. For example, one could imagine many queries want to select news stories on publicly traded companies. So in a stock monitoring application, many aggregate CQs will be defined on few indexes, with similar aggregate functions, but different joins and different windows. Thus, sharing can be expected to be heavy.

[2]Of course, the time span between each auction could just as easily be one week or one month. In Section 7, we discuss how our results can extend to a more general setting where each query may subscribe for a different time span.

EXAMPLE 1. *To make these concepts concrete, consider three queries (q_1, q_2 and q_3) that process two streams (s_1 and s_2) and are submitted to a DSMS with a capacity of 10 units. Figure 2(a) shows their query plan as generated by the query optimizer of the DSMS and is used to identify the shared operators. For example, operator A is shared by q_1 and q_2.*

As mentioned above, it is sufficient to abstract away the dependencies between operators of a CQ and retain only the information seen in Figure 2(b): the set of operators that comprise all the queries, the load of each operator, an indication of which queries each operator belongs to, and the user bids. We will use this representation in the rest of the paper for the readability of our figures.

3. RELEVANT BACKGROUND

A mechanism where users always maximize their payoff by truthfully revealing *all* their private information is called *strategyproof*. In many auction settings, clients' true valuations are the only private information, and hence in such settings *strategyproofness* means that clients maximize their payoff when bidding their true valuations. In this paper, we will refer to this property as *bid-strategyproofness*, as we also consider other private information: a user might conceivably lie about which operators are contained in her query, say by adding additional operators that are not part of the query she actually desires. In the context of our CQ admission auction then, we call a mechanism *strategyproof* when both bidding truthfully and submitting only the operators in the query actually desired by the user maximizes the user's payoff. In this work, all the mechanisms we propose are not only bid-strategyproof, but also strategyproof. Several standard auction problems are special cases of our auction problem for CQs.

Settings without Sharing. In the special case that there are no shared operators, the load of each query (which is the aggregate load of the query's operators) is the same, and there is room for k queries, then this is equivalent to the problem of auctioning k identical goods. Charging each of the k highest bidders the $(k+1)$st highest bid is well known to be bid-strategyproof. When $k = 1$, this is famously known as "Vickrey's second price" auction.

If CQs do not share operators, but the load of each query may be different, then the resulting problem is what is known in the literature as a Knapsack Auction problem, studied by Aggarwal and Hartline in [Aggarwal and Hartline 2006]. It is worth noting that in our CQ admission setting, the problem of maximizing total user valuation solves the well-known knapsack problem[3], and is thus NP-hard. It is this fact that tempers any hope of finding an efficient implementation of the standard, well-known, bid-strategyproof VCG mechanism: in order to calculate the correct payment for each winner, the VCG mechanism would require that we calculate an optimal solution to this NP-hard problem.

Operator Sharing. Operators shared between queries greatly complicate the task of the mechanism because the *profit density* of a query, which refers to the ratio of the bid for that query (or potential profit to be obtained from accepting the query) to the load of the query, depends on which other operators are selected. For example, consider a query q_i with low value and high load. In overload situations, query q_i would surely be rejected in a knapsack auction. But if all of q_i's operators were shared by high value queries, then the effective profit density of q_i (given that we know these high value queries were accepted) could be very high. This dependency between queries makes the mechanism's task much more complex in the case of our CQ auction than in the case of a knapsack auction. This complexity is illustrated by the fact that there is a polynomial time approximation scheme for finding the maximum value collection of items to

[3]In the knapsack problem, we are given n items, each with a weight and a value, and a knapsack with a given weight limit. The goal is to find a subset of the n items that maximizes total value while not exceeding the weight limit. If the operators in our problem are mapped to the knapsack items, their loads on the system are the weights of the knapsack items, and the user valuations are the values of the knapsack items, then any efficient solution for our problem will also efficiently solve the knapsack problem.

select in a knapsack auction, but even for a special case of our CQ auction, the densest subgraph problem, it is not known how to approximate the optimal solution to within a constant factor in polynomial time [Feige et al. 2001].

Characterizations of Strategyproofness. A CQ auction where the only private information is the amount each user values her query is called a single-parameter setting. In single-parameter settings, an allocation mechanism is called *monotone* if every winning bidder remains a winning bidder when increasing her bid. The *critical value* of user i is the value c_i where if the user bids higher than c_i, she wins, but if she bids lower than c_i, she loses. Note that the existence of a critical value for each user is guaranteed by the preceding monotonicity property. It is shown in [Nisan 2007] that a mechanism is bid-strategyproof if and only if it is both *monotone* and each winning user's payment is equal to her *critical value*.

One final auction setting related to our CQ auction is the *single minded bidders (SMB) auction problem*, studied by Lehmann et al [Lehmann et al. 2002]. Each single-minded bidder i is interested in a specific collection S_i from the set of items being auctioned (for a CQ auction the items being auctioned would be server capacity units and S_i would be the units needed to process the collection of operators in the query q_i). In addition to bid-strategyproofness. [Blumrosen and Nisan 2007; Lehmann et al. 2002] provide a characterization for strategyproofness in this setting that applies to our setting. Their characterization of a strategyproof mechanism for an SMB auction differs only slightly from the above characterization for bid-strategyproofness: the definition of monotonicity is expanded. In terms of our CQ admission auction, monotonicity means that not only must a winning bidder remain a winning bidder when increasing her bid, but also must remain a winner when submitting a query comprised of a strict subset of the operators in the admitted query.

Admission Control in DSMSs. Admission control in a DSMS has been studied extensively in the context of load shedding which is done at the tuple level with the primary objective of controlling the response time of queries during runtime [Gedik et al. 2005], [Tatbul et al. 2003] or satisfying the QoS requirements [Pham et al. 2011]. They do not consider strategyproofness or profit maximization. The work in [Wolf et al. 2009] and [Wolf et al. 2008] selects a subset of CQs to run every epoch in a similar fashion to our work. However, their goal is to maximize the utilization of the system and the overall importance of the CQs. In contrast, our work focuses on profit maximization, strategyproofness and sybil immunity even at the expense of system utilization.

4. PROPOSED MECHANISMS

In this section, we present several greedy CQ auction mechanisms. Each of these mechanisms has the following form: (1) sort queries in order of decreasing profit density (bid per unit of required server load), and then (2) admit queries until the server is full.

The intuition is that we wish to accept queries with high valuation to load ratio. Next we present these mechanisms in detail.

4.1 Clients Chosen by Remaining Load (CAR)

In order to set the stage, we start by describing a naïve approach that uses the remaining (i.e., additional) load of CQs for choosing winners and determining payment values. This approach accurately captures how much extra load each admitted CQ will contribute to the total load on the server. However, We show how this approach is not bid-strategyproof.

Consider the following natural mechanism using the aforementioned greedy scheme for choosing winners. The mechanism first chooses each winner based on a value we define called a query's "remaining load." Then the mechanism charges each winner a payment that also depends on that user's remaining load. We refer to the mechanism as the CAR mechanism (CQ Admission based on Remaining load). We will show that using such a payment scheme is not bid-strategyproof due to the fact that user's payments are dependent on their bids.

Algorithm 1 Our basic fair share mechanism (CAF).
Input: A set of queries with their static fair share loads C_i^{SF} and their corresponding bids b_i.
Output: The set of queries to be serviced and their corresponding payments.

(1) Set priority Pr_i to b_i/C_i^{SF} for each query i.
(2) Sort and renumber queries in non-increasing Pr_i so that $Pr_1 \geq Pr_2 \geq \ldots \geq Pr_n$.
(3) Add the maximal prefix of the queries in this ordered list that fits within server capacity to the winner list.
(4) Let *lost* be the index of the first losing user in the above priority list.
(5) Charge each winner i a payment of $p_i = C_i^{SF}(b_{lost}/C_{lost}^{SF})$. Charge all other users 0.

Selecting Winners. We sort the CQs in non-increasing order of priority Pr_i, where $Pr_i = b_i/C_i^R$ and C_i^R is defined as:

DEFINITION 3. (Remaining Load C_i^R) *The* remaining load C_i^R *of query i is equal to the total load of all the operators of q_i except those operators that are shared with CQs that have already been chosen as winners.*

In every iteration through the loop, the algorithm chooses the query with the highest priority and if there is enough remaining capacity in the system to accommodate it, places it in the set of winners. At the end of each iteration, the remaining loads C_i^R as well as the priorities of the yet-unchosen queries are updated. We demonstrate these steps with the example in Figure 2(b).

Calculating Payments. We naturally base our first payment mechanism on the known bid-strategyproof k-unit $(k+1)$th-price auction. Recall from Section 3 that a simple strategyproof mechanism for a k-unit auction is to charge each winning bidder the bid amount of the $(k+1)$th highest bidder. Hence, we define q_{lost} to be the CQ with highest priority that is not a winner. Then, the payment of each winning CQ q_i is calculated as follows: $p_i = C_i^R \cdot b_{lost}/C_{lost}^R$. If the query does not belong to the *winners* list, then the payment is zero.

Remaining Load Algorithm Applied to Example 1. The initial remaining loads of q_1 , q_2 and q_3 are 5, 6, and 10, respectively, and their priorities are 11,12 and 10. During the first iteration of the above algorithm, q_2 is chosen first. Since operator A is chosen as part of q_2, the remaining load of q_1 becomes the load of operator B (just 1 unit) and its priority becomes 55. Consequently, during the second iteration q_1 is chosen. The remaining capacity in the system is 3. During the third iteration, q_3 is chosen, however it does not fit in the remaining capacity in the system. As a result, the winners list is composed of q_1 and q_2, and q_{lost} is q_3. As a result, the payments for q_1 and q_2 is \$10 per unit load, which amount to respective payments of \$10 and \$60.

Strategyproofness. The above payment mechanism at first glance seems bid-strategyproof since it is based closely on the well-known bid-strategyproof second-price auction mechanism. However, it is not, since a winning user i who shares operators with other winning users can gain by bidding lower than her true value. She can strategically bid low enough so that she gets chosen for service *after* the users she shares operators with, but still high enough to win. This will result in a lower remaining load C_i^R and thus in a lower payment.

4.2 Clients Chosen by Static Fair Share Load (CAF, CAF+)

At this point it has become clear that using remaining load $\left(C_i^R\right)$ for setting payments of users is problematic because of the dependence of these values on the user's bid. Therefore, we consider using a fixed load that does not change over the course of the winner selection algorithm, and we use that same fixed load to calculate payments.

We define the *static fair share load* as follows.

Algorithm 2 Our aggressive fairshare mechanism (CAF+).
Input: A set of queries with their static fair share loads C_i^{SF} and their corresponding bids b_i.
Output: The set of queries to be serviced and their corresponding payments.

(1) Set priority Pr_i to b_i/C_i^{SF} for each query i.
(2) Sort and renumber queries in non-increasing Pr_i so that $Pr_1 \geq Pr_2 \geq \ldots \geq Pr_n$.
(3) For $i = 1 \ldots n$, add user i to the winner list if doing so does not exceed capacity.
(4) For each winner i, calculate $last(i)$, as defined in Definition 6.
(5) Charge each winner i a payment of
$p_i = C_i^{SF}(b_{last(i)}/C_{last(i)}^{SF})$. Charge all other users 0.

DEFINITION 4. *Let o_j be an operator that has a load of c_j and is shared among l different CQs, then the static fair share load of o_j per CQ is defined as $c_j^{SF} = c_j/l$. Hence, the* static fair share load *of a CQ q_i is defined as $C_i^{SF} = \sum_{o_j \in Q_i} c_j^{SF}$.*

In the following subsections we propose two bid-strategyproof payment mechanisms using the same greedy scheme, but based on static fair share load: CAF and CAF+.

CAF (CQ Admission based on Fair share). Our first bid-strategyproof mechanism that depends on the static fair share load as defined in Definition 4 is shown in Algorithm 1.

Selecting winners. Steps 1 through 3 of Algorithm 1 greedily select winners as follows. A priority is assigned to each operator, where the priority is the value-load ratio: $Pr_i = b_i/C_i^{SF}$. Then the list of CQs is sorted in descending order of these priority values. The algorithm admits CQs from the priority list in this order as long as the remaining load C_i^R of hosting the next CQ does not cause system capacity to be exceeded. (Note that the load considered while checking capacity constraints is not the static fair share load.) The algorithm stops as soon as the next CQ does not fit within server capacity.

Calculating payments. Once we have selected the winners, we calculate the payment for each winning user according to steps 4 and 5 of Algorithm 1.

CAF Applied to Example 1. Since q_1 shares operator A with q_2, C_1^{SF} is 3 and C_2^{SF} is 4. During the first iteration of CAF, the priorities of q_1, q_2 and q_3 are 18.34, 18, and 10. As a result, CAF chooses q_1 first and then q_2. Again, q_3 is q_{lost}. Thus the payments for q_1 and q_2 are \$10 per unit load, which amount to respective payments of \$30 and \$40.

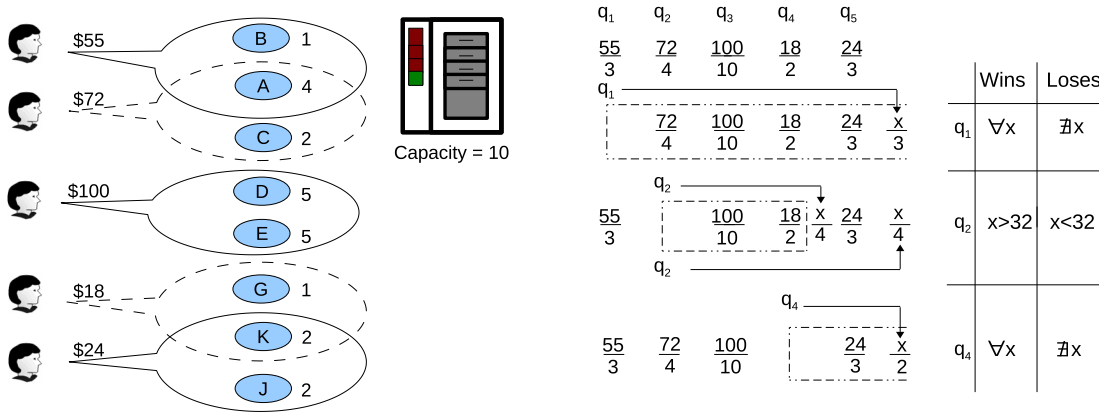
Strategyproofness. We prove the following theorems by using the characterization of bid-strategyproof mechanisms for any single-parameter setting (Section 3).

THEOREM 1. *The CAF mechanism is bid-strategyproof.*

PROOF. The CAF winner selection is clearly monotone: any winning bidder could not become a loser by increasing her bid since she will only move up in the priority list by doing so. The CAF payments are also equal to the users' critical values. If user i bids $b_i' < C_i^{SF}(b_{lost}/C_{lost}^{SF})$, then we would have $b_i'/C_i^{SF} < b_{lost}/C_{lost}^{SF}$ and we know that both user i and user *lost* could not fit together on the server with the other winners, so user i will become a loser. □

THEOREM 2. *The CAF mechanism is strategyproof.*

PROOF. This results from the fact that the characterization for SMB auctions in [Lehmann et al. 2002] carries over to our setting (see Section 3), and that CAF satisfies their additional monotonicity requirement that when a winning bidder asks for only a subset of the operators in her query, she still wins. □



(a) Queries in Example 1 in addition to two queries.

(b) The movement windows of the queries in Example 1 as determined by CAF+ (x is the lying bid).

Figure 3: Illustration of movement windows

CAF+: An Extension to CAF. CAF+ extends CAF by allowing the algorithm to continue until there are no unserviced CQs left that will fit in the remaining server capacity.

Selecting winners. While CAF stops as soon as it encounters a query whose load exceeds remaining capacity, CAF+ skips over any queries that are too costly, continuing onto more light-weight queries in the priority list. (See Algorithm 2.)

Calculating payments. In CAF+, every query that is selected to be serviced has a *movement window*. The algorithm calculates the payment of each selected query (winning user) based on each user's *movement window*. Intuitively, the movement window of a winning user is the amount of freedom the user has to bid lower than her actual valuation without losing.

DEFINITION 5. *A user's* movement window *is defined as a sublist of the complete list of queries ordered in descending priority* $Pr_i = b_i/C_i^{SF}$. *We will refer to this list as the* priority list. *The movement window of winning user* i *begins with the user just after user* i *in the priority list, and ends at the first user* j *in the priority list that satisfies the following property: if user* i*'s bid was changed so that it directly followed the position of user* j *in the priority list, CAF+ would no longer choose query* i *as a winner. If such a user* j *does not exist, then user* i*'s movement window spans the entire remainder of the priority list.*

DEFINITION 6. *For each winning query* q_i, $last(i)$ *is defined to be the first query which is outside* q_i*'s movement window. If there are no queries remaining outside the movement window of* q_i, *then* $last(i)$ *is set to null.*

The payment in CAF+ (Algorithm 2) is calculated for each query after the set of queries to be serviced is determined. For each winner i, the algorithm first calculates the identity of $last(i)$. Then the payment for the selected query is defined as $p_i = C_i^{SF} \cdot b_{last(i)}/C_{last(i)}^{SF}$. If user i's movement window included all remaining queries in the priority list, i.e., if $last(i) = null$, then the payment of user i is 0.

CAF+ Applied to Example 1. To demonstrate the difference between CAF and CAF+, assume two additional queries to those of Example 1, namely q_4 and q_5, with bids $b_4 = 18$ and $b_5 = 24$ respectively (See Figure 3(a)). Three operators of loads 1, 2 and 2 are needed by the two queries. Query q_4 consists of the first two operators (G and K) for a total load of 3, while q_5 consists of the latter two operators (K and J) for a total load of 4. Thus, C_4^{SF} is 2 and C_5^{SF} is 3. All the values of the other queries remain intact. During the first iteration of CAF+, the priorities of q_1

through q_5 are 18.34, 18, 10, 9 and 8. As a result, CAF+ chooses q_1 first and then q_2. Again, q_3 is not admitted because its remaining load exceeds the remaining system capacity. As opposed to CAF which would have stopped at this point, CAF+ continues to admit q_4 because its remaining load (3) fits in the available system capacity. Figure 3(b) shows the movement windows for the winning queries namely q_1, q_2 and q_4. The movement window for q_1 and q_4 spans the rest of the priority list. Thus, their payments are \$0. The payment for q_2, however, is \$8 per unit (\$32 for the whole query) because its movement window ends at q_5. This is because after choosing q_1, q_2's remaining load is 2, and it can lower it's priority by lowering it's price (variable x) down to a point where it is just selected. That point is right before q_5, thus its payment is determined by q_5's price per unit.

Strategyproofness. Under CAF and CAF+, a user might be tempted to lie about which operators are contained in her query because of the concept of fair share load. However, both CAF and CAF+ are *bid-strategyproof* (clients maximize their payoff when bidding their true valuations) and *strategyproof* (clients maximize their payoff when both bidding truthfully and submitting only the operators in the query actually desired by the user)

THEOREM 3. *The CAF+ mechanism is bid-strategyproof.*

PROOF. Proof of this theorem is included in the Appendix (Seciton A.1). □

THEOREM 4. *The CAF+ mechanism is strategyproof*

PROOF. As with CAF, we note that CAF+ is not only bid-strategyproof, but strategyproof. The reasoning is the same as for CAF (see Section 4.2). □

4.3 Clients Chosen by Total Load (CAT, CAT+)

We have developed two more mechanisms that are exactly analogous to the mechanism from Section 4.2, except that we replace every incidence of the static fairshare load C_i^{SF} with that total load $C_i^T = \sum_{o_j \in Q_i} c_j$. Thus we have two mechanisms.

—**CAT** (CQ Admission based on Total load): analogous to CAF described in Section 4.2.

—**CAT+**: analogous to CAF+ described in Section 4.2.

CAT Applied to Example 1. In example 1 C_1^T, C_2^T and C_3^T are 5, 6 and 10 units. Thus Pr_1, Pr_2 and Pr_3 are 11, 12, and 10. Consequently, CAT chooses q_1 and q_2 to be serviced. The payments for q_1 and q_2 are \$10 per unit load, which amount to respective payments of \$50 and \$60.

It is easy to verify that the proofs of bid-strategyproofness carry over to these modified versions of the algorithms and payments. We therefore have the following four theorems.

THEOREM 5. *The CAT mechanism is bid-strategyproof.*

THEOREM 6. *The CAT mechanism is strategyproof.*

THEOREM 7. *The CAT+ mechanism is bid-strategyproof.*

THEOREM 8. *The CAT+ mechanism is strategyproof.*

4.4 A Profit Guarantee

While we will experimentally show that the above greedy mechanisms perform quite well for profit maximization (Section 6), they do not admit *provable* profit guarantees that are reasonable (due to some special, pathological input instances). We thus investigate a basic mechanism that is based only on user bids rather than density: the CQs are simply sorted in decreasing bid order, and then selected from the top until the next CQ does not fit in system capacity. The chosen CQs then pay a price equal to the bid of the first losing CQ. We refer to this basic solution as the *Greedy-by-Valuation* (GV) mechanism. While GV also does not admit a profit guarantee, we propose a strategyproof randomized mechanism based on GV called *Two-Price*, that has a provable profit guarantee.

Specifically, Two-price is competitive (in expectation) with the best optimal constant pricing mechanism. A *constant pricing* mechanism as defined in [Aggarwal and Hartline 2006], is any mechanism (strategyproof or not) where the users are all charged the same price, call it p, and those who bid strictly higher than p are winners, those who bid strictly lower than p are losers, and those who bid equal to p may be designated winners or losers arbitrarily by the mechanism. Winners must all pay p and losers pay 0. *Profit* is defined to be the sum of the payments that the mechanism charges or receives from the users.

A *constant pricing* mechanism is *valid* if all winners fit within server capacity, and so we will only consider valid constant prices. *Optimal constant pricing profit* (OPT_C) then refers to the maximum possible profit that can be attained from any valid constant pricing mechanism (strategyproof or not). We choose to focus on constant pricing optimality in this paper because with the shared processing of queries in our problem, other standard profit benchmarks seem difficult to compete with. Two other natural profit benchmarks include optimal pricing per unit load and optimal monotone pricing, both of which generalize optimal constant pricing and were discussed in the context of Knapsack Auctions in [Aggarwal and Hartline 2006]. But because of our shared processing between queries, the processing load required of each query is not clear cut. Hence both proportional and monotone pricing definitions become problematic.

The Two-price Mechanism. We now show that by only using two distinct prices, under the assumption that the users all have distinct valuations, we are able to find a bid-strategyproof mechanism that approximates optimal constant pricing profit. We show however that there is a trade-off between the run-time of the mechanism and its profit. We first present a mechanism that runs in time exponential in the number of duplicate valuations, then explain how a polynomial time version of it gives a weaker profit guarantee. The algorithm is shown in Figure 3.

The first phase of the Two-price mechanism (Steps 1 and 2) follows our greedy scheme (using user valuations), the second phase (Step 3) is an exhaustive search that gives the potential exponential running time in terms of number of duplicate valuations, and the last phase (Steps 4 through 6) contains the randomization and is essentially identical to the Random Sampling Optimal Price auction of [Goldberg et al. 2006].

Note that in Step 3 of the mechanism we run an exhaustive search on all possible subsets of the critical set of queries with duplicate valuations. The possibility of sharing of server capacity between queries is what requires us to take this potentially time consuming step, as the problem of optimally determining which subset of queries to admit in the face of such sharing is a generalization of the well known NP-hard bin packing problem.

Strategyproofness: A randomized mechanism is *bid-strategyproof in expectation* if for every user i, the expected payoff for user i is maximized when user i bids her true valuation v_i [Nisan 2007]. A randomized mechanism is *bid-strategyproof in the universal sense* if it is not only bid-strategyproof in expectation, but it is also "ex post" bid-strategyproof. That is, regardless of the outcome of the randomness, users always maximize their payoff by bidding their true valuations. In other words, a universally bid-strategyproof randomized mechanism is a probability distribution over bid-strategyproof deterministic mechanisms [Nisan 2007].

To prove that our Two-price mechanism is bid-strategyproof in the universal sense, we depend

Algorithm 3 *Two-price mechanism.*
Input: Set of n queries and corresponding user valuations $v_1 \dots v_n$.
Output: Set of winners and their corresponding payments.

(1) Sort and renumber the queries in order of decreasing valuation, so $v_1 \geq v_2 \geq v_3 \geq \ldots \geq v_n$, breaking ties arbitrarily.
(2) Let H be the ordered set of queries that comprise the maximal prefix of queries from this sorted list that fits within our server capacity. Let L be the ordered set of losers (remaining queries not chosen for H) and let v_L be the valuation corresponding to the first query in L.
(3) If the last query in H has valuation v_L, the set of queries in H must be adjusted as follows. Let D be the set of all users with valuation equal to v_L, and let d be the cardinality of D. Let $H' = H - D$. Let D^* be the largest subset of D that fits within capacity along with H'. Redefine $H = H' + D^*$
(4) Partition the users from H evenly into two sets, A and B, uniformly at random. Renumber queries separately in each set as in step 1. I.e., $v_1 \geq v_2 \geq \ldots \geq v_a$ for the a queries in set A, and $v_1 \geq v_2 \geq \ldots \geq v_b$ for the b queries in set B, again breaking ties arbitrarily.
(5) Calculate the optimal constant price profit of each set of queries: $OPT(A) = \max_{i \in A} iv_i$ and $OPT(B) = \max_{i \in B} iv_i$. Let $k = \arg\max_{i \in A} iv_i$ and let $p_A = v_k$. Similarly, let $j = \arg\max_{i \in B} iv_i$ and let $p_B = v_j$.
(6) Use the price p_A to determine the winners from set B and use the price p_B to determine the winners from set A. Specifically, the winners from set B are those users whose valuations are greater than p_A, and these winners are each charged a payment of p_A. Similarly determine winners and payments for users in set A.

on some existing results. In [Aggarwal and Hartline 2006], *mechanism composition* is defined as follows.

Definition 7. *Define the composite mechanism $M_1 \circ M_2$, of two mechanisms M_1 and M_2, as:*

(1) Simulate M_1 and let H be the set of winners.
(2) Simulate M_2 on the set H.
(3) Offer a price to each winner of Step 2 that is the maximum of the price she is offered by M_1 and M_2.

They then define a mechanism to be *composable* if it is both bid-strategyproof and the set of chosen winners does not change as any winning user varies her bid above her critical value. Finally, they prove the following lemma.

Lemma 1. *([Aggarwal and Hartline 2006]) If mechanism M_1 is composable and mechanism M_2 is bid-strategyproof then the composite mechanism $M_1 \circ M_2$ is bid-strategyproof.*

Theorem 9. *The* Two-price *mechanism is bid-strategyproof.*

For the proof of the this theorem please see the Appendix (Section A.2).

Note that because the Two-price mechanism allocates winners and sets payments entirely independent of each query's load, it is not only bid-strategyproof, but strategyproof.

We now state the competitiveness of Two-price for profit maximization. We assume user valuations range from 1 to h and we use OPT_C to refer to the optimal constant pricing profit.

Theorem 10. *The expected profit of the* Two-price *mechanism is at least $OPT_C - 2h$.*

For the proof of the this theorem please see the Appendix (Section A.3).

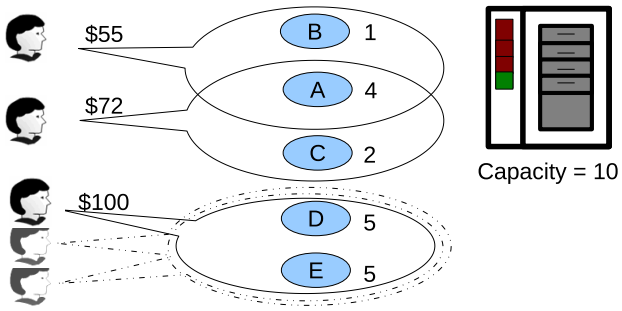


Figure 4: This figure illustrates the third user from Example 1 perpetrating a sybil attack by forging two additional fake users. The real queries are indicated in solid lines while the fake queries are indicated in dashed lines. The presence of these fake queries creates the illusion that user 3's operators are in higher demand, which could conceivably influence the mechanism to either charge the third user less, or service her when she would not have otherwise been serviced.

The next theorem applies to the polynomial-time mechanism that results when Step 3 of Algorithm 3 is omitted.

THEOREM 11. *The expected profit of the polynomial-time mechanism defined by the* Two-price *mechanism without Step 3 is at least* $OPT_C - dh$*, where* d *is the number of identical valuations in the input.*

For the proof of the this theorem please see the Appendix (Section A.4).

In this section we presented mechanisms that are strategyproof. Next, we investigate their sybil immunity.

5. SYBIL ATTACK

In this section we consider a strategic behavior that is well-known in the context of reputation systems like that of eBay and Amazon for rating sellers, buyers and products: a *sybil attack*. A user who behaves strategically using a sybil attack forges multiple ("fake") identities to manipulate the system. In reputation systems a user might try to boost the reputation of some entity by perhaps adding positive recommendations from false users [Friedman et al. 2007]. In our setting, a sybil attack amounts to creating false identities to submit additional queries that the user does not need or value in order to manipulate the mechanism (see Figure 4.)

Thus we define a mechanism to be *sybil immune* if a user can never increase her payoff by submitting additional fake, no-value queries. Sybil immunity is referred to as false-name proof in the context of General Vickrey Auctions [Sakurai et al. 1999].

We make the natural assumption that if a fake query is chosen to be serviced, the sybil attacker is responsible for making the fake query's payments, so a user's payoff is the aggregate payoff that she gains from the queries of all of her identities. We will show here that CAT is sybil immune, while the rest of the mechanisms are not. To the best of our knowledge, this is the first time that sybil immunity has been proposed in data management, and we note that the notion of sybil immunity can apply to any mechanism design problem.

DEFINITION 8. *We define a mechanism to be* vulnerable *to sybil attack if there exists an input instance where there is a user who can increase her payoff by perpetrating a sybil attack.*

DEFINITION 9. *We define a mechanism to be* universally vulnerable *to sybil attack if in every input instance, every user has a way to improve her payoff by perpetrating a sybil attack.*

Table I: An example of a sybil attack that beats CAT+.

User	1	2	"3"
v_i	100	89	$100\epsilon + \epsilon$
C_i^T	1	0.9	ϵ
Pr_i	100	< 100	> 100
Round 1	100	< 100	picked
Round 2	exceeds cap.	picked	picked
Payments p_i	0	0	$\leftarrow 100\epsilon$
Payoffs	0	$89 - 100\epsilon$	N/A

DEFINITION 10. *We say that a mechanism is* immune to sybil attack *if for every input instance, no user can increase her payoff by perpetrating a sybil attack (i.e., it is not vulnerable). We also use the term* sybil immunity *to refer to this property.*

5.1 Attacks Against the Fair Share Mechanisms

Unfortunately, our proposed fair-share schemes of Section 4.2 are vulnerable to sybil attack. A user i can employ the following strategy using a sybil attack to improve her payoff: simply create fake users with negligible valuations whose queries share operators with q_i. A sybil attack of this kind will lower the attacker's fair share load, improving her ranking and enabling her to be selected as a winner while simultaneously decreasing her payment to an affordable amount. Note that it is always possible for the attacker to set her fake users' valuations low enough so that they are not in danger of being selected as winners, and hence will require no additional payment from the attacker.

Indeed, we can prove that in *any* given instance of the CQ admission problem, *any* user can gain from employing a sybil attack against our fair share mechanism.

THEOREM 12. *Both the CAF or CAF+ mechanisms are* universally vulnerable *to sybil attack.*

5.2 Attacks Against the Total Load Mechanisms

In contrast to this vulnerability of our fair share mechanisms, the total load payment mechanisms (CAT and CAT+), described in Section 4.3, seem at first to be robust to such attacks. While we have seen that a user's fair share can easily be reduced by creating fake identities, a user's total load is not dependent on the number of other users sharing her load, and therefore CAT and CAT+ should not (at least at first glance) be prone to such sybil attack strategies.

However, one of our total load mechanisms is not immune to sybil attack. We begin by giving the following characterization of sybil immunity. A mechanism is sybil immune if and only if both of the following properties hold:

(1) The arrival of additional queries will never cause a loser to become a winner with positive payoff.
(2) If the arrival of additional queries reduces a winner's payment by δ, the additional queries that become winners must be charged a total of at least δ by the mechanism.

We now show that CAT+ is vulnerable to sybil attack because it does not satisfy the above property 1.

THEOREM 13. *For the CQ admission problem, CAT+ is vulnerable to sybil attack.*

To see why, consider the example in Table I, in which a sybil attacker defeats our total load algorithm, CAT+. User 2 is a sybil attacker, creating a fake query that appears to the system as "user 3". Here, ϵ represents an arbitrarily small positive value. In this example, if user 2 does not perpetrate the attack, user 1 will get chosen for service, and then server capacity will be

reached, so user 2 would not get serviced. Whereas when user 2 introduces the fake "user 3," she is able to trick the system into choosing her instead of user 1. While user 2 is responsible for the fake user's payment, user 2 carefully created "user 3" so that its payment would be a negligible amount. Note that user 2's payment for query 2 is 0 since there is no one left after she is chosen.

Note that in this kind of sybil attack, the danger for user 2 (the attacker) is that when the fake "user 3" was chosen for service, user 2 had to make user 3's payment. Hence user 3's fake valuation and fake load had to be carefully chosen by user 2 so that user 2 found paying user 3's fee worthwhile. (Recall from Section 4.3 that payment of a winning user i is calculated as $C_i^T v_{lost}/C_{lost}^T$, so in our example, that makes $p_3 = 100\epsilon$). In this particular instance, user 2 had no payment of her own to pay because there are no users that have lower priority than user 2. This makes paying "user 3"'s payment affordable to user 2.

The good news is: our total load mechanisms are not always bad. First, while our fair share mechanisms are *universally vulnerable* to attack, there are instances under the total share mechanism that are robust to sybil attack. Second, and more notably, the CAT mechanism is immune to sybil attack. Thus far in our discussion of sybil attacks, we have been considering sybil attack in isolation from bid-strategyproofness. However, it is possible that a user can use a sybil attack *in conjunction* with lying about her valuation in order to increase her payoff. This possibility raises the question of whether adding sybil attacks to each user's set of possible strategies has removed our mechanism's bid-strategyproofness.

It turns out that our CAT mechanism remains bid-strategyproof even if we allow sybil attacks, *and* it remains immune to sybil attack, even if we allow users to lie about their valuations.

Definition 11. *We define a mechanism to be* sybil-strategyproof *if no user can improve her payoff by either lying about her valuation, perpetrating a sybil attack, or doing both simultaneously.*

We now give a characterization of sybil-strategyproof mechanisms. A mechanism is sybil-strategyproof if and only if both of the following properties hold:

(1) It is bid-strategyproof.
(2) The arrival of additional users (e.g., via a sybil attack) cannot decrease anyone's critical value by an amount more than the total payment charged to the additional users.

The above characterization is used to prove that CAT is sybil-strategyproof.

Theorem 14. *For the CQ admission problem, the CAT mechanism is sybil-strategyproof.*

5.3 Attacks Against the Randomized Mechanism

Our randomized *Two-price* mechanism, however, is not immune to sybil attack. This fact is proven by showing that the mechanism violates property 2 of our characterization of sybil immunity: a winning user can reduce her payment (in expectation) by introducing fake queries such that the fake queries incur less expected total charges than the amount her payment was reduced by.

Theorem 15. *The* Two-price *mechanism is vulnerable to sybil attack.*

Finally, we note that even if we modify Step 4 of the mechanism so that each query is placed in set A or B based on independent coin flips (so that H may not be evenly partitioned), the mechanism is still vulnerable to sybil attack. Again, the vulnerability is due to a violation of property 2 of our characterization of sybil immunity. Consider the instance where user 1 has valuation b and n_c users (which get placed into H along with user 1) all have a valuation of $c < b$. Set sizes for the users in H so that server capacity is exactly filled.

User 1 creates a fake user with valuation $d = c + \epsilon$, with size equal to the combined size of all the users with valuation c, kicking them out of H. While before user 1 was charged c with probability $1 - (1/2)^{n_c}$ and 0 with probability $(1/2)^{n_c}$, now that only user 1 and the fake user

Table II: Workload Characteristics

Num of workload sets	50
Num of queries	2000
Num of operators	$700 \sim 8800$
Max Degree of Sharing	$[1 - 60]$ - Zipf, skewness: 1
Max Bid	100 - Zipf, skewness: 0.5
Max Operator Load	10 - Zipf, skewness: 1
System Capacity	5K-10K-15K-20K

are in H, user 1 and her fake user is charged 0 with probability 1/2, and d with probability 1/2. For choice of ϵ that ensures $d/2 < c(1 - (1/2)^{n_c})$, user 1's expected payoff has decreased.

6. EXPERIMENTAL EVALUATION

In this section, we demonstrate the behavior of our proposed auction-based admission control mechanisms using simulation. First we present the experimental setup. Then we discuss the results.

6.1 Experimental Platform

In this section we describe all the technical characteristics of our experimental evaluation.

Mechanisms. We implemented all the proposed admission control mechanisms, as well as GV (Greedy by Valuation mechanism described in Section 4.4) in Java. We categorize the mechanisms into two groups: the density-based mechanisms (CAF, CAF+, CAT, and CAT+), which include all mechanisms that sort the queries based on the ratio of the bid to the load, and the valuation-based mechanisms (GV and Two-price) that sort the users' queries based on their bids.

Metrics. For each mechanism, we measured the following performance metrics:

—*Profit:* the sum of the payments of the admitted queries.

—*Admission rate:* The percentage of queries admitted.

—*Total user payoff:* the sum of the valuations (bids) of the admitted queries minus the payments. Total user payoff can be seen as an indication of total user satisfaction under each mechanism.

—*System utilization:* the used capacity of the server.

—The *runtime* for each mechanism.

The reported results are the average of running each algorithm on 50 different sets of workloads. Note that, for clarity, our figures do not show GV as it mirrors the behavior of Two-price in all experiments.

Workload. We summarize the workload parameters in Table II. We generated 50 sets of workloads for four different system capacities. Each set contains a number of different input instances. An input instance consists of users' queries along with their bids, and is parameterized by:

—*System capacity.*

—*Maximum degree of sharing:* The degree of sharing of an operator is the number of queries that share a single operator, and the maximum is taken over all the operators.

We varied the maximum degree of sharing from 1 to 60. We kept the average query load the same throughout a workload set, while varying the maximum degree of sharing. To achieve this, we generate a workload with the highest maximum degree of sharing (i.e., 60) and then gradually split the operators of the highest degree and distribute the resulting operators into other varying degrees within a workload. Each input instance consists of 2000 queries and between 700 and 8800 operators (the number of operators decreases as the degree of sharing increases).

The bids of each query are randomly generated according to a Zipf distribution with maximum bid value set to 100 and skewness parameter set to 0.5. We chose a skewed distribution (Zipf) to reflect the real-life scenario, where users' bids are correlated and close to each other, except for few out-of-range bids.

The load of each operator is also randomly generated according to a Zipf distribution with the maximum operator load set to 10 units and skewness parameter set to 1. Operators are assigned to queries randomly, where for each operator, the number of queries sharing it is drawn from a Zipf distribution with skewness parameter set to 1 and the maximum degree of sharing varying from 1 to 60. Again, this skewness of operator sharing reflects the real-life scenarios where there are few "hot" streams, where many users are monitoring via CQs, while many other streams are of less interest, and hence are less shared.

6.2 Experimental Results.

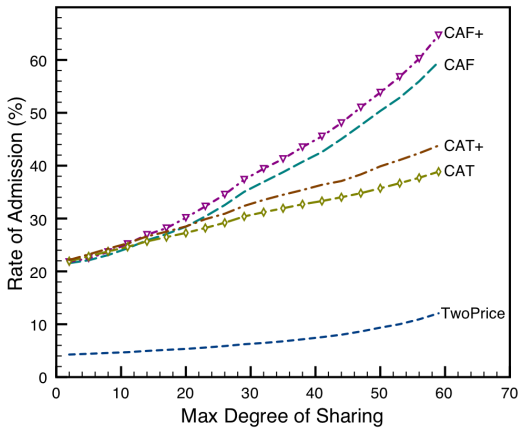
In this section we present the results of our experimental evaluation for the proposed mechanisms. In the first set of experiments we compare the behavior of all mechanisms in terms of system profit, user payoff and query admission rate. In the second set of experiments we present the results of a sensitivity analysis on the system capacity. Finally we compare the runtime performance of the algorithms.

Mechanism Comparison (Fig. 5)

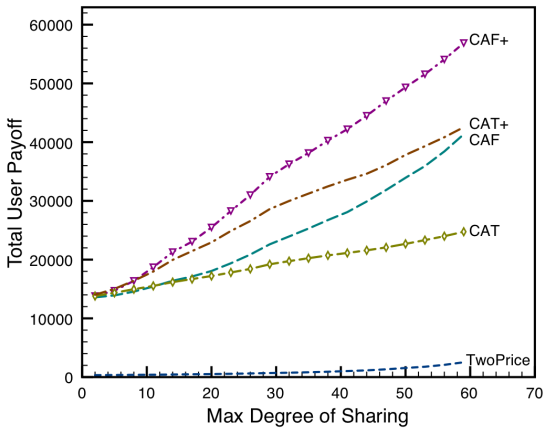
Figure 5(a) shows the percentage of admitted queries as the degree of sharing ranges from 1 to 60, for a system with capacity 15,000. All mechanisms admit more queries as the degree of sharing increases. This is due to the fact that the mechanisms are able to take advantage of the shared processing between queries, so more queries can be serviced using the same system capacity. Two-price always admits a smaller percentage of the queries than the density-based mechanisms (CAF, CAF+, CAT, CAT+) because it chooses queries by user bid only, without regard to query load.

Interestingly, profit (in Figure 5(c)) does not follow the same trend. CAF and CAT are the best for profit, as they do not admit queries as greedily as CAF+ and CAT+ do, which means the prices they charge for admitted queries are much higher than those of CAF+ and CAT+. The profit of CAF+ and CAT+ decreases as the degree of sharing increases because they are simply admitting so many queries (as sharing increases) that the prices they are charging for admitted queries continues to be driven downward. Due to the fact that queries are selected in decreasing order of density and charged a per-unit price equal to the per-unit bid of the first losing query, very few queries means higher prices, more queries means lower prices. The Two-price mechanism provides profit that consistently improves as the degree of sharing increases because its profit is close to the optimal constant pricing profit, which only improves as the number of queries that can fit within capacity increases. At the point where Two-price crosses over CAF and CAT (with degree of sharing near 53), we observe the same phenomenon that caused decreasing profit in CAF+ and CAT+. At the crossover point, CAF and CAT begin to admit such a high rate of queries that the prices they are charging are being driven dramatically downward (remember, query valuations are drawn from a skewed distribution), reducing overall profit faster than the gain in profit from admitting more queries. The profit of CAF in particular begins to really dive, as the payments are an increasing function of each query's fair share load, which also shrinks as the degree of sharing increases.

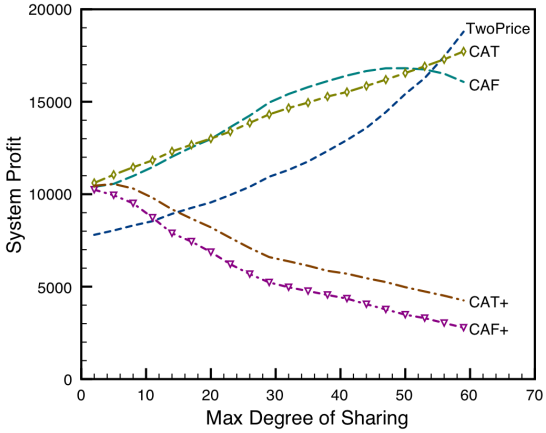
With respect to maximizing total user payoff (Figure 5(b)), the density based mechanisms always perform better than Two-price because they are able to admit more queries and satisfy more customers. CAF+, of course, has the highest payoff because not only are the most queries admitted under CAF+, but users are only paying for their fair share load, rather than for their total load. As the degree of sharing increases, CAF begins to overtake CAT+ in total user payoff because fair share load per user is decreasing, which decreases payments, increasing payoffs. Each query's total load on the other hand, remains constant as the degree of sharing increases.



(a) System Capacity = 15,000



(b) System Capacity = 15,000



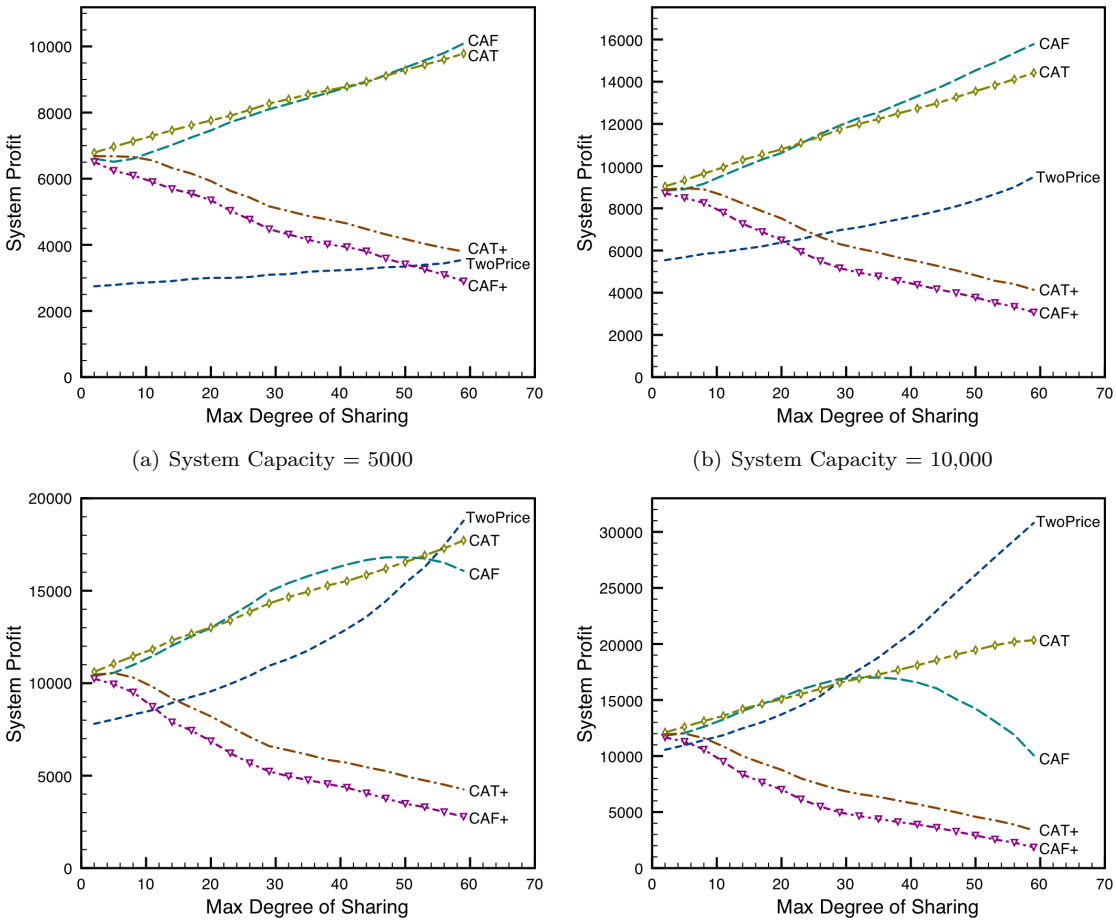
(c) System Capacity = 15,000

Figure 5: Figure 5(a) shows the percentage of queries serviced under each mechanism. Figure 5(b) shows total user payoff, which can be interpreted as a measure of total user-satisfaction. A user's payoff is defined as her valuation minus her payment. Seen here is the sum of winning users' payoffs. Figure 5(c) shows the system profit. System capacity is set to 15,000.

In terms of utilization, we found that all proposed mechanisms admit queries so as to utilize more than 98 percent of the system capacity, except for Two-price which utilizes between 96 percent and 98 percent. Note that system utilization is always above 96%, regardless of the profit and the admission policy. This shows that a good admission control policy gives higher profits for the same amount of resources.

Sensitivity to system capacity (Fig. 6)

In Figure 6, we show the system profit for three other system capacities. As system capacity increases, it is apparent that the crossover points (between CAF+, CAT+ and Two-price and between CAF, CAT and Two-price) are shifted to the left, to lower degrees of sharing. Indeed, as capacity increases, the picture as a whole seems to shift and scale downward to the lower end of max degree of sharing. When system capacity is close to the total query demand and the degree of sharing is high, the Two-price mechanism clearly overtakes all the density mechanisms for highest profit. As described above, this is due to the fact that so many of the queries are being serviced by the density mechanisms, driving down the prices being charged.



(a) System Capacity = 5000

(b) System Capacity = 10,000

(c) System Capacity = 15,000

(d) System Capacity = 20,000

Figure 6: System profit as system capacity varies from 5000 to 20,000.

Runtime performance (Table III)

We list the average runtime performance of each mechanism over all workloads with 2000 queries and capacity 15K in Table III. As a baseline, we also implemented a randomly admitting algorithm, which picks queries at random and stops at the first query that does not fit in the remaining capacity. The algorithms ran on an Intel Xeon 8 core 2.3GHz, with 16GB of RAM. The mechanisms only utilized one core. It is clear that the more aggressive mechanisms (CAF+ and CAT+) cannot scale compared to the simple ones. We note here that even though the density based mechanisms' runtime is only three to seven times more than the baseline random algorithm, they provide strategyproofness, and moreover CAT also provides sybil-immunity.

Table III: Runtime performance averages (in msecs) for each algorithm on 50 workloads with 2000 queries

Random	0.92
GV	2.003
Two-price	3.72
CAF	7.088
CAF+	12555.5
CAT	7.26
CAT+	10091.2

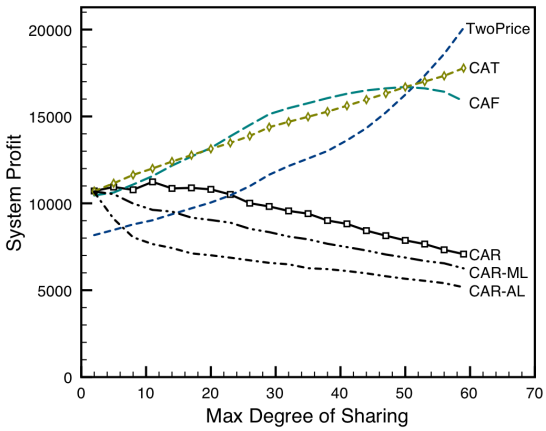


Figure 7: Profit of strategyproof mechanisms (CAF, CAT, and Two-price) vs the non-strategyproof CAR (when no user lies), CAR-ML(Moderate Lying workload) and CAR-AL(Aggressive Lying workload). System capacity = 15,000.

Manipulation of the System (Fig. 7)

Finally, we evaluate CAR for profit both in a setting where users are being truthful about their valuations, and in a setting where they strategize and bid less than their true valuations (i.e., "lie"). Since CAR is the only mechanism that is not strategyproof, such lying under CAR is to be expected.

To simulate strategizing users, we add an alternative bid to each client, which represents a lower bid than her valuation, and it is the product of her query valuation (original bid) and a lying factor. If a user's query shares many operators with other queries, she would strategize by bidding lower than her valuation thus lowering her payment and increasing her payoff. Therefore, if the ratio of *Static Fair Share/Total Load* is less than a certain threshold, the client will lie (i.e., submit the alternative bid) with a certain probability. We generated two workloads: a moderately lying workload and an aggressively lying workload. In the moderately lying workload, the threshold is set to 0.25, the probability of lying to 0.5, and the lying factor to 0.5, while in the aggressively lying workload, they are set to 0.35, 0.7 and 0.3 respectively.

Figure 7 shows the profit for three strategyproof mechanisms, CAF, CAT and Two-price, along with three different representations of the profit of CAR: CAR when no user lies, CAR-ML (CAR running the Moderate Lying workload) and CAR-AL (CAR running the Aggressive Lying workload). We see that when some users lie, the system profit decreases, motivating the need and desire of the system for a strategyproof mechanism. The profit of the three strategyproof mechanisms is dependable, while the profit from CAR is manipulable.

System Profit vs. User payoff tradeoff (Table IV)

The tradeoff between the System Profit to the Total User Payoff can be quantified using their ratio. Table IV shows this ratio for all the mechanisms at the degree of sharing of 40 where CAF provides the best system profit and at 60 where CAT provides the best system profit. Ideally this ratio should be equal to 1, since a mechanism like that would provide a fair balance between system profit and user payoff, giving both and equal weight. Clearly, CAT provides the best tradeoff between profit and user payoff, since it is the one closest to the ideal case. Two-price, on the other hand, is biased towards System Profit.

Table IV: Ratio of system profit to total user payoff

Degree of sharing	60	40
CAF+	0.04	0.1
CAT+	0.09	0.17
CAF	0.37	0.59
CAT	0.73	0.71
Ideal	1	1
Two-price	7.16	12.8

Table V: Properties of our proposed auction mechanisms.

Mechanism	Strategy-proof	Sybil Immune	Profit Guarantee	Admiss Rate	User Payoff	Profit
CAF	✓	×	×	High	Med	High
CAF+	✓	×	×	High	High	Low
CAT	✓	✓	×	Med	Med	High
CAT+	✓	×	×	Med	High	Low
Two-price	✓	×	✓	Low	Low	Med

7. DISCUSSION

Table V summarizes the desirable characteristics [4] of each mechanism alongside its performance for various metrics like profit maximization, total user payoff, and rate of admission.

To extend the proposed solutions to the more general setting of different queries wanting different minimum subscription lengths, we propose the following. Assume without loss of generality that the minimum subscription lengths the system wishes to offer are one day, one week, one month, and one year. Let each of these lengths be referred to as a subscription category. Partition system resources so that an appropriate fraction of total system capacity is allocated to each subscription category. For the queries in each category, run the strategyproof auction mechanism of your choice (see Table V) with the amount of system capacity allotted to that category. At the end of each day, reclaim the system capacity from those whose subscriptions expire that day and iterate: partition the currently remaining available system resources among the different categories of subscriptions and again run a separate auction mechanism for each category.

The good news is that because each auction is being run independently and separately, and all our auctions are bid-strategyproof, this scheme as a whole is still bid-strategyproof. However, introducing these repeated rounds of auctions introduces a new type of potential strategic behavior. Under such a scheme, users may not be honest about the subscription periods they are interested in. For example, a user who wants to run a CQ for one month in July may instead bid for a two month subscription starting in June if she believes demand is low enough in June to get charged a sufficiently low price that paying for two months is cheaper than paying for one month starting in July. Guarding against this sort of strategic behavior in addition to maintaining bid-strategyproofness would be a challenging problem for future work.

Another issue to consider is the energy consumption of the DSMS center. Different levels of system operation incur different energy costs. This can be coupled with the observation that it might be more profitable not to fully utilize the available capacity. Indeed, this is what our experiments clearly suggest. Hence, an extension is to decide what is the most beneficial capacity for a given auction, while considering both the profit as well as the savings from energy reduction.

[4] Admission Rate, Total User Payoff, and Profit are in terms of relative performance as degree of sharing increases. For Profit, in the special case that degree of sharing is high and system capacity is almost as high as total system demand, the profit from CAF and CAT begins to dwindle and the profit from Two-price is highest.

8. CONCLUSIONS

This work sits at the intersection of two different lines of data management research, namely user-centric data management and data stream management, and utilizes techniques from the domain of game theory. By using an auction model, we are able to explore a novel way of describing user preferences in the CQ admission control problem. Although most data stream admission control (load shedding) algorithms work at the tuple level, we believe that focusing on the query level, as we do in this work, is equally important.

We provided a model for the problem that allows us to establish its difficulty and complexity. We introduced the notion of *sybil immunity* for auction mechanisms and designed greedy and randomized auction mechanisms for this problem which are all *strategyproof*. We conducted experiments to evaluate the performance of these mechanisms for metrics such as profit, admission rate, and total user payoff, and we showed that one of the mechanisms namely CAT is sybil immune.

Our results show that CAT and CAF are the best mechanisms to use for profit maximization in most circumstances. However, if you have a high degree of operator sharing, and your system capacity is close to the total demand of the queries requesting service, then Two-price performs better for profit maximization. As expected, the greedy mechanisms (CAF, CAF+, CAT, and CAT+) provide better admission rate and payoff than Two-price. CAF+ and CAT+ are best for total user payoff, while CAF and CAF+ have the highest query admission rate as the degree of sharing increases.

Finally, it is worth mentioning that our proposed auction-based admission control mechanisms can be used in conjunction with traditional (i.e., non-continuous) query processing where there are many queries that share processing, and executed concurrently.

Acknowledgments

This research was supported in part by an IBM faculty award (Kirk Pruhs), pre-doctoral Andrew Mellon Fellowship (Shenoda Guirguis), and from NSF grants CNS-0325353, CCF-0514058, IIS-0534531, IIS-0746696, CCF-0830558, IIS-1050301, CCF-1115575 and CNS-1253218.

We would like to thank our colleagues from University of Pittsburgh, Adam Lee, Thao Pham and Alex Connor, as well as the Algorithmic Game Theory researchers, namely Avrim Blum, Gerhard Woeginger, Nikhil Bansal, Maxim Sviridenko and Ho-Leung Chan for their helpful comments.

REFERENCES

Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. 2003. Aurora: a new model and architecture for data stream management. *VLDBJ 12,* 2, 120–139.

Aggarwal, G. and Hartline, J. D. 2006. Knapsack auctions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. SODA '06. ACM, New York, NY, USA, 1083–1092.

Al Moakar, L., Chrysanthis, P. K., Chung, C., Guirguis, S., Labrinidis, A., Neophytou, P., and Pruhs, K. 2010. Admission control mechanisms for continuous queries in the cloud. In *Proceedings of the 26th IEEE International Conference on Data Engineering*. ICDE'10. IEEE Computer Society, Long Beach, CA, U.S.A, 409–412.

AmazonEC 2009. Amazon elastic compute cloud, http://aws.amazon.com/ec2/.

Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., and Widom, J. 2003. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin 26,* 1, 19–26.

Babcock, B., Babu, S., Motwani, R., and Datar, M. 2003. Chain: operator scheduling for memory minimization in data stream systems. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. SIGMOD '03. ACM, New York, NY, USA, 253–264.

Blumrosen, L. and Nisan, N. 2007. Combinatorial auctions. In *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, Eds. Cambridge University Press, Chapter 11, 267–300.

Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul,

N., and Zdonik, S. 2002. Monitoring streams: a new class of data management applications. In *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB '02. VLDB Endowment, 215–226.

Chakravarthy, S. and Jiang, Q. 2009. *Stream Data Processing: A Quality of Service Perspective - Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Advances in Database Systems, vol. 36. Kluwer.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S. R., Reiss, F., and Shah, M. A. 2003. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. SIGMOD '03. ACM, New York, NY, USA, 668–668.

Coral8 2004. http://www.coral8.com/.

Douceur, J. R. 2002. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. Springer-Verlag, London, UK, 251–260.

Esper Tech 2006. http://esper.codehaus.org.

Feige, U., Peleg, D., and Kortsarz, G. 2001. The dense k-subgraph problem. *Algorithmica 29,* 3, 410–421.

Fiat, A., Goldberg, A. V., Hartline, J. D., and Karlin, A. R. 2002. Competitive generalized auctions. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. STOC '02. ACM, New York, NY, USA, 72–81.

Friedman, E., Resnick, P., and Sami, R. 2007. Manipulation-resistant reputation systems. In *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, Eds. Cambridge University Press, Chapter 27, 267–300.

Gedik, B., Wu, K.-L., Yu, P. S., and Liu, L. 2005. Adaptive load shedding for windowed stream joins. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. CIKM '05. ACM, New York, NY, USA, 171–178.

Goldberg, A. V., Hartline, J. D., Karlin, A. R., Saks, M., and Wright, A. 2006. Competitive auctions. *Games and Economic Behavior 55*, 242–269.

Krishnamurthy, S., Wu, C., and Franklin, M. 2006. On-the-fly sharing for streamed aggregation. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. SIGMOD '06. ACM, New York, NY, USA, 623–634.

Lehmann, D., Oćallaghan, L. I., and Shoham, Y. 2002. Truth revelation in approximately efficient combinatorial auctions. *J. ACM 49,* 5, 577–602.

Madden, S., Shah, M., Hellerstein, J. M., and Raman, V. 2002. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. SIGMOD '02. ACM, New York, NY, USA, 49–60.

Microsoft StreamInSight 2008. http://www.microsoft.com/sqlserver/2008/en/us/r2-complex-event.aspx.

Nisan, N. 2007. Introduction to mechanism design. In *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, Eds. Cambridge University Press, Chapter 9, 267–300.

Pham, T. N., Moakar, L. A., Chrysanthis, P. K., and Labrinidis, A. 2011. Dilos: A dynamic integrated load manager and scheduler for continuous queries. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops*. ICDEW '11. IEEE Computer Society, Washington, DC, USA, 10–15.

Sakurai, Y., Yokoo, M., and Matsubara, S. 1999. A limitation of the generalized vickrey auction in electronic commerce: robustness against false-name bids. In *AAAI/IAAI*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 86–92.

Sellis, T. K. 1988. Multiple-query optimization. *ACM Transactions on Database Systems 13,* 1, 23–52.

Sharaf, M. A., Chrysanthis, P. K., Labrinidis, A., and Pruhs, K. 2008. Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Transactions on Database Systems 33,* 1, 1–44.

Streambase 2006. http://www.streambase.com.

System S 2008. http://domino.research.ibm.com/comm/research_projects.nsf/pages/esps.index.html.

Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., and Stonebraker, M. 2003. Load shedding in a data stream manager. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*. VLDB '03. VLDB Endowment, 309–320.

Wolf, J., Bansal, N., Hildrum, K., Parekh, S., Rajan, D., Wagle, R., and Wu, K.-L. 2009. Job scheduling strategies for parallel processing. Springer-Verlag, Berlin, Heidelberg, Chapter Job Admission and Resource Allocation in Distributed Streaming Systems, 169–189.

Wolf, J., Bansal, N., Hildrum, K., Parekh, S., Rajan, D., Wagle, R., Wu, K.-L., and Fleischer, L. 2008. Soda: An optimizing scheduler for large-scale stream-based distributed computer systems. In *Middleware 2008*, V. Issarny and R. Schantz, Eds. Lecture Notes in Computer Science, vol. 5346. Springer Berlin / Heidelberg, 306–325.

A. PROOFS OF STRATEGYPROOFNESS

A.1 Proof of Theorem 3

The CAF+ mechanism is bid-strategyproof.

PROOF. The CAF+ winner selection is monotone: any winning bidder could not become a loser by increasing her bid since she will only move earlier in the priority list by doing so. The CAF+ payments, by definition of each user's movement window (Definition 5) are precisely equal to the minimum value the user must bid in order to remain a winner. □

A.2 Proof of Theorem 9

The Two-price *mechanism is bid-strategyproof in the universal sense.*

PROOF. We use Lemma 1. Let M_1 be the mechanism defined by steps 1 and 2 of the *Two-price* mechanism along with its corresponding critical payments: each user pays an amount equal to v_L, the highest losing bid as defined in Step 2 of the algorithm. This is bid-strategyproof as it is equivalent to a standard k-Vickrey auction (see Section 3), and it is composable since any winner varying her price above the highest losing bid does not change the set of winners. We let M_2 be the mechanism defined by the remaining steps of the Two-price mechanism. Note that M_2 is a *randomized bid-independent auction* (as defined in [Fiat et al. 2002]). Theorem 2.1 in [Fiat et al. 2002] states that an auction is universally bid-strategyproof if and only if it is bid-independent. Because the payments set by M_2 will always be higher than those set by M_1, we can invoke Lemma 1 to conclude our entire mechanism is bid-strategyproof. □

A.3 Proof of Theorem 10

The expected profit of the Two-price *mechanism is at least* $OPT - 2h$.

PROOF. Let $n(S, p)$ refer to the number of users in set S whose valuations are p or higher. Then the Two-price mechanism's expected profit can be expressed as $TP = \mathbf{E}[n(A, p_B)p_B + n(B, p_A)p_A]$. Observe that

$$\mathbf{E}[n(A, p_B)p_B] \geq \mathbf{E}[n(B, p_B)p_B] - p_B \geq \mathbf{E}[n(B, p^*)p^*] - p_B,$$

where p^* is the optimal constant price if our input was the set H and the second inequality holds by definition of p_B. Then observe that

$$\mathbf{E}[n(B, p^*)p^*] = \frac{n(H, p^*)p^*}{2} = \frac{OPT(H)}{2} = \frac{OPT}{2} \tag{1}$$

where OPT(H) refers to the optimal solution if the input is only the queries in H, and the last equality holds because any *valid* optimal constant price can be no less than the minimum valuation of any user in H. Putting these together and upper bounding p_B with h gives us

$$\mathbf{E}[n(A, p_B)p_B] \geq \frac{OPT - 2h}{2}.$$

By symmetric arguments for $\mathbf{E}[n(B, p_A)p_A]$ and linearity of expectation we can then conclude $TP \geq OPT - 2h$. □

A.4 Proof of Theorem 11

The expected profit of the polynomial-time mechanism defined by the Two-price *mechanism without Step 3 is at least* $OPT - dh$, *where* d *is the number of identical valuations in the input.*

PROOF. We can show that eliminating Step 3 of the *Two-price* mechanism, yielding a polynomial-time algorithm, gives a weaker profit guarantee parameterized by the maximum number of duplicate valuations. Specifically, if there are d valuations in the input that are identical, then the profit guarantee decreases to $OPT - dh$. The analysis is similar to the proof of Theorem 10, the difference being that we can no longer claim $OPT(H) = OPT$ as in equation (1). Eliminating

Step 3 of the mechanism means we can only say that H has at least 1 of the queries in the set D, while OPT has at most $d-1$ of the queries in D, so we instead obtain $OPT(H) \geq OPT-(d-2)h$. Combining this with the rest of the analysis, which remains the same, gives us the following result. □

B. SYBIL ATTACK PROOFS

B.1 Proof of Theorem 12

Both the CAF or CAF+ mechanisms are universally vulnerable *to sybil attack.*

Proof. Consider any given user i. We have two cases: either user i is a winner under the mechanism in question (either CAF or CAF+), or i is a loser.

Case 1. If i is a loser, then i can gain by perpetrating a sybil attack as follows. Let j be a winning user such that $j = \arg\max_k Pr_k$. Choose the number s of forged queries to be such that $v_i/(C_i^T/s) > v_j/C_j^{SF}$. User i creates s fake users, each with query identical to q_i. Doing so makes $C_i^{SF} \leq C_i^T/s$, which by choice of s means $Pr_i = v_i/C_i^{SF} > v_j/C_j^{SF} = Pr_j$. Since j was the winner with highest priority, and all users have total load at most 1 by assumption, i is now a winner. We can see that i's payoff has improved because when i was a loser her payoff was 0, while now it is $v_i - p_i = v_i - C_i^{SF} v_j/C_j^{SF} > 0$. User i can also ensure that she makes no payments for the fake queries she created by setting their valuations to be sufficiently small so that they all lose.

Case 2. If i is a winner, then i can gain by using a sybil attack to reduce i's payment, p_i. If i simply creates one fake user whose query is identical to q_i, C_i^{SF} will be reduced, which reduces $p_i = C_i^{SF}(v_{lost}/C_{lost}^{SF})$. Again, user i chooses the fake user's valuation to be a sufficiently small value to ensure that the fake query's priority is low enough to not get serviced. □

B.2 Proof of Theorem 14

For the CQ admission problem, the CAT mechanism is sybil-strategyproof.

Proof. Property 1 of the characterization of sybil-strategyproofness is satisfied since we have already seen from Theorem 5 that CAT is bid-strategyproof. To satisfy property 2 of the characterization, it is sufficient to show that adding additional users to the instance does not decrease any user's critical value. Consider any user i. In the CAT mechanism, user i has critical value $p_i = C_i^T(v_{lost}/C_{lost}^T)$. (Recall that *lost* is defined to be the user with highest priority not selected to be serviced by CAT.) Since the arrival of additional users cannot change C_i^T, we need only show that the arrival of additional users cannot decrease $v_{lost}/C_{lost}^T = Pr_{lost}$.

We proceed by induction on the number of additional users that arrive. Assume inductively that introducing the first k users cannot reduce Pr_{lost}. Define $lost(k)$ to be *lost* in the instance that includes only the first k fake users, and let $lost(k+1)$ be *lost* in the instance that includes the first $k+1$ users. We must show that $Pr_{lost(k+1)} = v_{lost(k+1)}/C_{lost(k+1)}^T \geq Pr_{lost(k)} = v_{lost(k)}/C_{lost(k)}^T$.

Consider user j, the $(k+1)$th newly arriving user. If user j has priority $Pr_j \leq Pr_{lost(k)}$, then $lost(k+1) = lost(k)$ and hence $Pr_{lost(k+1)} = Pr_{lost(k)}$. If $Pr_j > Pr_{lost(k)}$ then $Pr_{lost(k+1)} \geq Pr_{lost(k)}$.

Hence the critical value of user i cannot be decreased by the arrival of additional users. □

B.3 Proof of Theorem 15

The Two-price *mechanism is vulnerable to sybil attack.*

Proof. Consider the following input instance. User 1 has valuation $b > 1$ and users 2 and 3 have valuation $c > 1$, where $b > c$ and c is an integer. Suppose all three users make it past the first three Steps of the mechanism into the set H with some positive capacity to spare. (Assume all other users have very large size and valuation less than 1 and were thus placed in set L.)

The mechanism will always charge user 1 a price of c, regardless of the randomness, giving user 1 a payoff of $b - c$. However, user 1 can benefit from sybil attack as follows. User 1 creates

$2c-3$ fake queries, each with valuation $1+\epsilon$, and each with size small enough that they are also placed in the set H. We consider three cases.

Case 1: users 2 and 3 are in the opposite partition from user 1. Without loss of generality, assume user 1 is in set A and users 2 and 3 are in set B. In this case, it is impossible for the fake users to change the payoff of user 1: $c-2$ of them are placed in the set B, but $(1+\epsilon)c < 2c$, so p_B remains at c as before. And none of the fake users are winners.

Case 2: users 2 and 3 are in opposite partitions from one another. Without loss of generality, assume users 1 and 2 are placed in set A and user 3 is placed in set B. Now, since $(1+\epsilon)c > c$, the price p_B becomes $1+\epsilon$, giving user 1, our sybil attacker, a lower price. The fake users in set A also must pay $1+\epsilon$, but there are only $c-2$ of them, so user 1's net payoff becomes $b-(1+\epsilon)-(c-2)(1+\epsilon) > b-c$, for small enough ϵ.

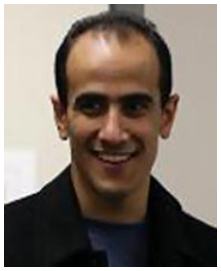
Case 3: users 1, 2 and 3 are in the same partition. Assume they are all placed in set A. Then user 1's new price is again $1+\epsilon$. Again, even after making the payments for her fake users, users 1's payoff has improved.

Thus, overall, user 1's expected payoff improves due to her sybil attack. □

Lory Al Moakar is a member of the Advanced Data Management Technologies Laboratory and a Ph.D. candidate at the Department of Computer Science at the University of Pittsburgh. While there, she received the 2007 and 2012 Orrin E. and Margaret M. Taulbee award for Excellence in Computer Science. She received her B.S. in Computer Science in 2005, from the American University of Science and Technology in Zahle, Lebanon. After her graduation, she worked as a part-time instructor at Computer Science Department at the American University of Science and Technology in Zahle, Lebanon. Her research interests include Data Stream Management Systems, scheduling and query optimization. She is currently working under the supervision of Professors Alexandros Labrinidis and Panos K. Chrysanthis on class-based scheduling in Data Stream Management Systems.

Panos K. Chrysanthis is a Professor of Computer Science, and founder and co-director of the Advanced Data Management Technologies Laboratory, University of Pittsburgh. He is also an Adjunct Professor at Carnegie Mellon University and at the University of Cyprus. His lab has a broad focus on user-centric data management for scalable network-centric applications and has fostered interdisciplinary collaborations between computer science, medicine, astronomy, and materials science. His research contributions in principles, algorithms and prototype systems have been documented in more than 150 papers in top database journals and prestigious, peer-reviewed data management conferences and workshops and have appeared in textbooks. In 1995, he received one of the first NSF CAREER Awards and in 2010, he was recognized as a Distinguished Scientist by ACM. In 2007, he also became a Senior Member of IEEE. Dr. Chrysanthis has served on the editorial board of several journals, as a PC member in all major data management conferences, and as a General and Program Chair of a number of conferences and workshops. He was invited to offer tutorials, contribute book chapters, and organize and participate in NSF planning meetings. Dr. Chrysanthis received his B.S. degree (Physics with concentration in Computer Science, 1982) from the University of Athens, Greece. He earned the M.S. and Ph.D. degrees (Computer and Information Sciences, 1986 and 1991) from the University of Massachusetts at Amherst.

Christine Chung is the Jean C Tempel Assistant Professor of Computer Science at Connecticut College. She received her Ph.D. in Computer Science from University of Pittsburgh in 2009 under the supervision of Kirk Pruhs. While there, she earned the 2008 Orrin E. and Margaret M. Taulbee award for Excellence in Computer Science. She earned her M.A. in Mathematics Education from Teachers College, Columbia University in 2003, and her M.Eng. (2000) and B.A. (1999) degrees in Computer Science from Cornell University. In 2001, due to her work with "K-Zone" on ESPN's Sunday Night Baseball, she earned an Emmy Award from the National Academy of Television Arts and Sciences for Innovative Technical Achievement. Her current research interests are in algorithms and algorithmic game theory.



Shenoda Guirguis is a Software Performance Engineer (R&D) at Intel Co where he analyzes and fine tunes the performance of Oracle Server Software products on new Intel hardware products. Before joining Intel, he received his Ph.D. in Computer Science - Data Management from University of Pittsburgh. He also received a Masters degree from University of Pittsburgh and another from Alexandria University. He has a total of ten publications, two patents as results from his summer research internships at NEC Labsa America and GoldenGate Co. (now part of Oracle). He received seven awards and fellowships. Shenoda is passionate about Data Management research and innovation. His areas of interest include data streams, cloud data management, multi-tenant databases, and web-databases.

Alexandros Labrinidis received his Ph.D. degree from the University of Maryland, College Park in 2002, and M.S. and B.S. degrees from the University of Crete, Greece in 1995 and 1993 respectively. He is currently an associate professor at the Department of Computer Science of the University of Pittsburgh and co-director of the Advanced Data Management Technologies Lab. He is also an adjunct associate professor at Carnegie Mellon University (CS Dept.). Dr. Labrinidis' research focuses on user-centric data management for network-centric applications, including web-databases, data stream management systems, sensor networks, and scientific data management (with an emphasis on big data). He has published over 60 papers at peer-reviewed journals, conferences, and workshops; he is the recipient of an NSF CAREER award in 2008. Dr. Labrinidis is currently the Secretary/Treasurer for ACM SIGMOD, and has served as the Editor of SIGMOD Record, and in numerous program committees of international conferences/workshops.

Panayiotis Neophytou is a member of the Advanced Data Management Technologies Laboratory and Ph.D. candidate at the Computer Science Department at the University of Pittsburgh. He received his B.S. in Computer Science in 2003, from the University of Cyprus and his MS n Computer Science in 2011, from the University of Pittsburgh. After receiving his B.S., he worked as a Research Assistant for a year at the Pervasive Computing Lab of the CS Department of the University of Cyprus. His research interests include Data Management, Data Streams Processing and Systems Integration in distributed environments. He is currently working under the supervision of Professors Panos K. Chrysanthis and Alexandros Labrinidis, on a Continuous Workflows Computation Model and prototype, to leverage stream data processing and the traditional workflow processing under a single system.

Kirk Pruhs received a bachelors in mathematics and computer science from Iowa State University in 1984, and a Ph.D. in computer science in 1989 from the University of Wisconsin, where his adviser was Udi Manber. He has been a faculty member in the computer science department at the University of Pittsburgh since 1989, and he currently holds the rank of professor. His main research interests are in algorithmic problems related to green computing, energy and thermal management, resource management, and scheduling. He was an organizer of the NSF workshop on the Science of Power Management, and the Dagstuhl series of seminars on Scheduling. He is chairman of the steering committee of the Workshop on Models and Algorithms for Planning and Scheduling Problems. He is on the editorial boards of ACM Transactions on Algorithms, INFORMS Journal of Computing, Journal of Scheduling and Sustainable Computing: Informatics and Systems.